Complete Unrestricted Backtracking Algorithms for Satisfiability

Inês Lynce and João Marques-Silva
Department of Information Systems and Computer Science,
Technical University of Lisbon,
IST/INESC/CEL, Lisbon, Portugal
{ines,jpms}@sat.inesc.pt

Abstract

In recent years, different backtrack search Propositional Satisfiability (SAT) algorithms have proposed relaxing the identification of the backtrack point in the search tree. Even though relaxing the identification of the backtrack point can be significant in solving hard instances of SAT, it is also true that the resulting algorithms may no longer be *complete*. This paper proposes a new backtrack search strategy, *unrestricted backtracking*, that naturally captures relaxations of the identification of the backtrack point in the search tree, most notably search restarts and random backtracking. Moreover, the paper proposes a number of conditions that guarantee the completeness of generic unrestricted backtracking SAT algorithms.

Introduction

Propositional Satisfiability is a well-known NP-complete problem, with theoretical and practical significance, and with extensive applications in many fields of Computer Science and Engineering, including Artificial Intelligence and Electronic Design Automation.

Current state-of-the-art SAT solvers incorporate sophisticated pruning techniques as well as new strategies on how to organize the search. Effective search pruning techniques are based, among others, on nogood learning and dependency-directed backtracking (Stallman & Sussman 1977) and backjumping (Gaschnig 1979), whereas recent effective strategies introduce variations on the organization of backtrack search. Examples of such strategies are weak-commitment search (Yokoo 1994), search restarts (Gomes, Selman, & Kautz 1998) and random backtracking (Lynce, Baptista, & Marques-Silva 2001).

Advanced techniques applied to backtrack search SAT algorithms have achieved remarkable improvements (Bayardo Jr. & Schrag 1997; Marques-Silva & Sakallah 1999; Moskewicz *et al.* 2001), having been shown to be crucial for solving hard instances of SAT obtained from real-world applications. Moreover, and from a practical perspective, the most effective algorithms are complete, and so able to prove what local search is not capable of, i.e. unsatisfiability. Indeed, this is often the objective in a large number of significant real-world applications.

Nevertheless, it is also widely accepted that local search (Selman & Kautz 1993) can often have clear advantages with respect to backtrack search, since it is allowed to start the search over again whenever it gets *stuck* in a locally optimal partial solution. This advantage of local search has motivated the study of approaches for relaxing backtracking conditions (while still assuring completeness). The key idea is to *unrestrictedly* choose the point to backtrack to, in order to avoid thrashing during backtrack search. Moreover, one can think of combining different forms of relaxing the identification of the backtrack point. In this paper, we propose a new generic framework for implementing different backtracking strategies, referred to as *unrestricted backtracking*. Besides describing the unrestricted backtracking search strategy, we also establish completeness conditions for the resulting SAT algorithms.

The remainder of this paper is organized as follows. The next section presents definitions used throughout the paper. Afterwards, we briefly survey backtrack search SAT algorithms. Then we introduce the unrestricted backtracking search strategy and analyze examples of specific formulations of unrestricted backtracking. In addition, we relate completeness conditions with the different forms of backtracking. Finally, we describe related work, and conclude by suggesting future research directions.

Definitions

This section introduces the notational framework used throughout the paper. Propositional variables are denoted $x_1, \ldots, x_n$, and can be assigned truth values 0 (or F) or 1 (or T). The truth value assigned to a variable x is denoted by $\nu(x)$. (When clear from context we use $x = \nu_x$, where $\nu_x \in \{0, 1\}$). A literal l is either a variable x_i or its negation $\neg x_i$. A clause ω is a disjunction of literals and a CNF formula φ is a conjunction of clauses. A clause is said to be *satisfied* if at least one of its literals assumes value 1, *unsatisfied* if all of its literals assume value 0, *unit* if all but one literal assume value 0, and *unresolved* otherwise. Literals with no assigned truth value are said to be *free literals*. A formula is said to be *satisfied* if all its clauses are satisfied, and is *unsatisfied* if at least one clause is unsatisfied. A *truth assignment* for a formula is a set of assigned variables and their corresponding truth values. The SAT problem consists of deciding whether there exists a truth assignment to the variables such that the formula becomes satisfied.

SAT algorithms can be characterized as being either *complete* or *incomplete*. Complete algorithms can establish unsatisfiability if given enough CPU time; incomplete algorithms cannot. In a search context, complete algorithms are often referred to as *systematic*, whereas incomplete algorithms are referred to as *non-systematic*.

Backtrack Search SAT Algorithms

Over the years a large number of algorithms have been proposed for SAT, from the original Davis-Putnam procedure (Davis & Putnam 1960), to recent backtrack search algorithms (Bayardo Jr. & Schrag 1997; Zhang 1997; Marques-Silva & Sakallah 1999; Moskewicz *et al.* 2001) and to local search algorithms (Selman & Kautz 1993), among many others.

The vast majority of backtrack search SAT algorithms build upon the original backtrack search algorithm of Davis, Logemann and Loveland (Davis, Logemann, & Loveland 1962). The backtrack search algorithm is implemented by a *search process* that implicitly enumerates the space of 2^n possible binary assignments to the n problem variables. Each different truth assignment defines a *search path* within the search space. A *decision level* is associated with each variable selection and assignment. The first variable selection corresponds to decision level 1, and the decision level is incremented by 1 for each new decision assignment [1]. In addition, and for each decision level, the *unit clause rule* (Davis & Putnam 1960) is applied. If a clause is unit, then the sole free literal must be assigned value 1 for the formula to be satisfied. In this case, the value of the literal and of the associated variable are said to be *implied*. The iterated application of the unit clause rule is often referred to as Boolean Constraint Propagation (BCP).

In chronological backtracking, the search algorithm keeps track of which decision assignments have been toggled. Given an unsatisfied clause (i.e. a *conflict* or a *dead end*) at decision level d, the algorithm checks whether at the current decision level the corresponding decision variable x has already been toggled. If not, the algorithms erases the variable assignments which are implied by the assignment on x, including the assignment on x, assigns the opposite value to x, and marks decision variable x as toggled. In contrast, if the value of x has already been toggled, the search backtracks to decision level $d - 1$.

Recent state-of-the-art SAT solvers utilize different forms of non-chronological backtracking (Bayardo Jr. & Schrag 1997; Marques-Silva & Sakallah 1999; Moskewicz *et al.* 2001), in which each identified conflict is analyzed, its causes identified, and a new clause created to explain and prevent the identified conflicting conditions. Created clauses are then used to compute the backtrack point as the *most recent* decision assignment from all the decision assignments represented in the recorded clause. Moreover, some of the (larger) recorded clauses are even-

[1] Observe that all the assignments made before the first decision assignment correspond to decision level 0.

tually deleted. Clauses can be deleted opportunistically whenever they are no longer *relevant* for the current search path (Marques-Silva & Sakallah 1999).

Unrestricted Backtracking

Unrestricted backtracking relaxes the condition that backtracking must be taken to the *most recent* decision assignment in a recorded clause. In other words, whenever a dead-end is reached, the search algorithm is allowed to *unrestrictedly* backtrack to any point (i.e. decision level) in the current search path. Clearly, this unrestricted backtrack step can also be the usual chronological or non-chronological backtrack steps. Besides the freedom for selecting the backtrack point in the decision tree, unrestricted backtracking entails a *policy* for applying different backtrack steps in sequence. Each backtrack step can be selected among chronological backtracking (CB), non-chronological backtracking (NCB) or alternative forms of backtracking (AFB) (e.g., search restarts, weak-commitment search, random backtracking, heuristic backtracking, or constant-depth backtracking, among many others). More formally, unrestricted backtracking consists of defining a sequence of backtrack steps $\{\mathrm{BSt}_1, \mathrm{BSt}_2, \mathrm{BSt}_3, \ldots\}$ such that each backtrack step BSt_i can either be a chronological, a non-chronological or an alternative form of backtracking, i.e. $\mathrm{BSt}_i \in \{\mathrm{CB}, \mathrm{NCB}, \mathrm{AFB}\}$.

The definition of unrestricted backtracking (UB) allows capturing the backtracking search strategies used by current state-of-the-art SAT solvers.

Clearly, if the UB strategy specifies always applying the CB step or always applying the NCB step, then we respectively capture the chronological and non-chronological backtracking search strategies.

For example, consider a UB strategy consisting of applying an AFB step after every K chronological backtrack steps, and where the AFB step is a search restart. Then this UB strategy corresponds to search restarts in (chronological) backtrack search (Gomes, Selman, & Kautz 1998) [2], hence being an an incomplete algorithm. If instead of chronological backtracking, non-chronological backtracking with clause recording is applied in between AFB steps, and if the number of conflicts in between AFB steps increases by a constant value i_K, then the resulting UB strategy corresponds to search restarts with non-chronological backtrack search (and with clause recording) (Baptista & Marques-Silva 2000).

If instead of a search restart the AFB step in the previous strategies consists of a random backtrack step, then the resulting algorithm corresponds to stochastic systematic search (Lynce, Baptista, & Marques-Silva 2001).

Moreover one can also envision new, though more elaborate, UB strategies, that involve different forms of AFB. An example of such a UB strategy could be applying an AFB corresponding to random backtracking after every K conflicts (where K can be strictly increasing), and applying an AFB step corresponding to a search restart after every M conflicts (where M can also be strictly increasing).

Why Unrestricted Backtracking?

In the previous section we illustrated how unrestricted backtracking captures the most successful backtracking strategies currently used for SAT. Nevertheless, and besides allowing capturing different backtracking strategies, one may wonder the actual usefulness of unrestricted backtracking.

We start by observing that a unified representation for different backtracking strategies allows establishing general completeness conditions for *classes* of backtracking strategies and not only for each individual strategy, as it has often been done. By utilizing a unified representation, we can establish conditions that apply to *all* variations of unrestricted backtracking. This will naturally simplify establishing completeness results for future backtracking strategies.

It should also be observed that the most competitive SAT solvers, e.g. (Marques-Silva & Sakallah 1999; Moskewicz *et al.* 2001) do indeed apply multiple backtracking strategies. Hence, unrestricted backtracking naturally models the organization of modern state-of-the-art SAT solvers.

[2] In this case we assume that the variable selection heuristic can be randomized as described in (Gomes, Selman, & Kautz 1998).

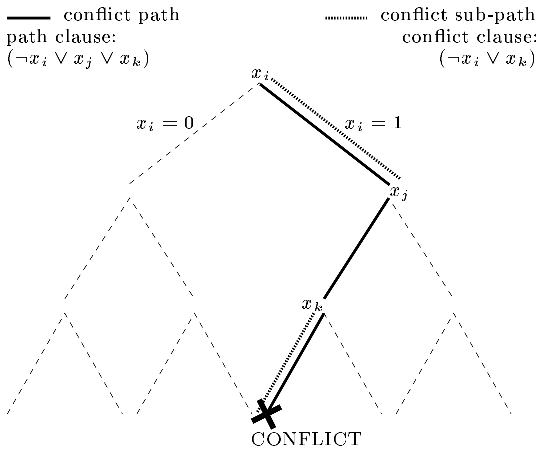

Figure 1: Search tree definitions

Completeness Issues

In this section we address the problem of guaranteeing the completeness of algorithms that implement some form of unrestricted backtracking. As illustrated in the previous sections, unrestricted backtracking can yield incomplete algorithms. Before proceeding, however, we need to introduce a few definitions.

Preliminaries

In what follows we assume the organization of a backtrack search SAT algorithm as described earlier in this paper. The main loop of the algorithm consists of selecting a variable assignment (i.e. a *decision assignment*), making that assignment, and propagating that assignment using BCP. In the presence of an unsatisfied clause (i.e. a *conflict*) the algorithm backtracks to a decision assignment that can be toggled [3]. Each time a conflict is identified, all the current decision assignments define a *conflict path* in the search tree. (Observe that we restrict the definition of conflict path solely with respect to the decision assignments.) After a conflict is identified, we may apply a *conflict analysis* procedure (Bayardo Jr. & Schrag 1997; Marques-Silva & Sakallah 1999; Moskewicz *et al.* 2001) to identify a subset of the decision assignments that represent a sufficient condition for producing the same conflict. This subset of decision assignments is in general represented as a new clause, being referred to as a *conflict-induced clause* (being also known as a *nogood*, or simply as a *conflict-clause*). The subset of decision assignments that is declared to be associated with a given conflict is referred to as a *conflict sub-path*. A straightforward conflict analysis procedure consists of constructing a clause with *all* the decision assignments in the conflict path. In this case the created clause is referred to as a *path-clause*. Figure 1 illustrates these definitions.

As a final note, for the backtracking strategies considered below, we restrict ourselves to the ones specifically used in the SAT domain. As a result, we do not consider non-chronological backtracking strategies that are not based on recording conflict-induced clauses.

Standard Backtracking Strategies

It is well-known that the chronological and non-chronological backtracking algorithms proposed for SAT are complete (Davis & Putnam 1960; Marques-Silva & Sakallah 1999). Basically these algorithms are complete because there is always an implicit explanation for why a solution cannot be found in the portion of the search space already searched; either the set of already toggled variables or previously recorded conflict clauses. Also, observe that non-chronological backtracking is complete even when clause deletion is applied.

Besides the completeness results, it is also useful to understand whether CB and NCB can repeat conflict paths and conflict sub-paths.

Theorem 1 *For an unrestricted backtracking algorithm that only implements either chronological backtrack (CB) steps or non-chronological backtrack (NCB) steps, the following holds:*

- *For either CB or NCB, no conflict paths are repeated.*
- *For CB conflict sub-paths can be repeated.*
- *For NCB, where all recorded clauses are kept, no conflict sub-paths are repeated.*
- *For NCB where some (large) recorded clauses are opportunistically deleted, conflict sub-paths can be repeated.*

[3] Without loss of generality, we assume that NCB also uses variable toggling.

Since the search space is implicitly enumerated in a given order, and since the search process only toggles untoggled variables, we are guaranteed never to repeat a conflict path. This is true for both CB and NCB. Moreover, observe that this is true for NCB even when clauses get opportunistically deleted. Indeed, a recorded clause can only be deleted provided some other recorded clause explains *why* a portion of the search space does not contain a solution (Marques-Silva & Sakallah 1999). A simple induction argument allows establishing completeness and consequently the fact that conflict paths are not repeated.

Clearly, conflict sub-paths can be repeated for CB, since no clause recording takes place. Moreover, and for NCB, if all conflict-clauses are kept, then no conflict sub-paths can be repeated, since the search process and BCP prevent sets of decision assignments that directly unsatisfy clauses. Finally, if recorded clauses can be deleted, then conflict sub-paths can be repeated, in particular conflict sub-paths associated with recorded clauses that get deleted during the search process.

In the next sections we analyze completeness conditions when AFB steps are also taken during the search process. We should emphasize that in all cases, the results established do *not* depend on the actual AFB step that is taken, but only on what is sufficient to be done to guarantee completeness in the presence of the AFB steps.

Strong Completeness Conditions

In this section we consider unrestricted backtracking algorithms that record (and keep) a clause for each identified conflict. In this situation, and given the definitions given above for a conflict path (and possibly associated path-clause or conflict-induced clause), the following results can be established.

Theorem 2 *An unrestricted backtracking algorithm does not repeat conflict paths provided it records a* path-clause *for each identified conflict.*

Since a path-clause captures the decision assignments associated with a conflict, the existence of that clause and the application of BCP guarantees that the same set of decision assignments becomes disallowed by the operation of the search algorithm. Clearly, in this case the algorithm does not loop and will eventually finish.

Corollary 3 *An unrestricted backtracking algorithm is complete provided it records a* path-clause *for each identified conflict.*

The previous argument can also be used for establishing the following results.

Theorem 4 *An unrestricted backtracking algorithm does not repeat conflict sub-paths provided it records a* conflict-clause *for each identified conflict.*

The same reasoning that was used above for path-clauses applies in this case. The existence of the conflict-clause and the application of BCP guarantees that the same set of decision assignments becomes disallowed by the operation of the search algorithm.

Corollary 5 *An unrestricted backtracking algorithm does not repeat conflict paths provided it records a* conflict-clause *for each identified conflict.*

Since the algorithm does not repeat conflict sub-paths, it must necessarily not repeat conflict paths. Moreover, and as above, completeness of the search algorithm is guaranteed.

Corollary 6 *An unrestricted backtracking algorithm that records a* conflict-clause *for each identified conflict is complete.*

Given the above results, one can further relate algorithms that record path-clauses and algorithms that record conflict-clauses. Basically, an algorithm that records conflict-clauses does not repeat conflict paths, but an algorithm that records path-clauses can repeat conflict sub-paths.

Theorem 7 *An unrestricted backtracking algorithm that records a* path-clause *for each identified conflict can repeat conflict sub-paths.*

Observe that path-clauses do not constrain proper sub-sets of decision assignments, and so these sub-sets may occur again during the search process.

The previous conditions for ensuring completeness entail recording a clause for each identified conflict. Hence, the number of clauses grows linearly with the number of conflicts, and so in the worst-case exponentially with the number of variables.

Finally, we note that the observation that keeping all recorded clauses yields a complete algorithm has been previously stated by others (Ginsberg 1993; Yokoo 1994), in the context of specific variations on backtracking algorithms.

Weak Completeness Conditions

The results established in the previous section guarantee completeness at the cost of recording (and keeping) a clause for each identified conflict. In this section we propose and analyze conditions for relaxing this requirement. We allow for some clauses to be deleted during the search process, and only require that some specific recorded clauses are kept. (We should note that clause deletion does not apply to chronological backtracking strategies, and also that in non-chronological strategies existing deletion policies do not compromise the completeness of the algorithm.) Afterwards, we propose other conditions that do not require specific recorded clauses to be kept. As described earlier, we assume that unrestricted backtracking consists of an arbitrary sequence of CB, NCB and AFB steps. Moreover, we say that a recorded clause is *kept* provided it is prevented from being deleted during the subsequent search.

Theorem 8 *An unrestricted backtracking algorithm is complete provided it records (and keeps) a* conflict-clause *for each identified conflict for which an AFB step is taken.*

We know that both chronological and non-chronological backtracking yield complete algorithms, even when large clauses are deleted in non-chronological backtracking. Moreover, if we record a conflict clause each time an AFB step is taken, then this same conflict sub-path will not be further repeated again during the remaining search, and so AFBs will subsequently be taken on *different* conflict sub-paths. Hence, the search process must necessarily terminate. Note that we can strengthen the previous result.

Theorem 9 *Given a integer constant M, an unrestricted backtracking algorithm is complete provided it records (and keeps) a* conflict-clause *after every M identified conflicts for which an AFB step is taken.*

The same reasoning used above applies. Instead of recording a conflict clause after every conflict for which an AFB step is taken, we record a clause after every M conflicts for which an AFB is taken.

Corollary 10 *Under the conditions of Theorem 8 and Theorem 9, the number of times a conflict path or a conflict sub-path is repeated is upper-bounded.*

As shown earlier, the resulting algorithm is complete. Hence, the number of different decision assignments considered is upper-bounded, and the same necessarily holds for the number of times a conflict sub-path or a conflict-path is repeated.

As one final remark, observe that for the previous conditions, the number of recorded clauses grows linearly with the number of conflicts where an AFB step is taken, and so in the worst-case exponentially in the number of variables.

Other approaches to guarantee completeness involve increasing the value of some constraint associated with the search algorithm. The following results illustrate these approaches.

Theorem 11 *Suppose an unrestricted backtracking strategy that applies a sequence of backtrack steps. If for this sequence the number of conflicts in between AFB steps strictly increases after each AFB step, then the resulting algorithm is complete.*

Observe that since the number of conflicts in between AFB steps is strictly increasing, then eventually the search algorithm will have a sufficient number of chronological or non-chronological backtrack steps to either prove satisfiability or unsatisfiability. We should also note that this result can be viewed as a generalization of the completeness-ensuring condition used in search restarts, that consists of increasing the backtrack cutoff value after each search restart (Baptista & Marques-Silva 2000) [4]. Finally, observe that in this situation the growth in the number of clauses can be made polynomial, provided clause deletion

[4] Observe that, given this condition, the resulting algorithm resembles iterative-deepening.

is applied on clauses recorded from non-chronological backtrack steps.

The next result establishes conditions for guaranteeing completeness whenever large recorded clauses (due to an AFB step) are opportunistically deleted. The idea is to increase the size of recorded clauses that are kept after each AFB step. Another approach is to increase the life-span of large-recorded clauses, by increasing the relevance-based learning threshold (Bayardo Jr. & Schrag 1997).

Theorem 12 *Suppose an unrestricted backtracking strategy that applies a specific sequence of backtrack steps. If for this sequence, either the size of the largest recorded clause or the size of the relevance-based learning threshold is increased after each AFB step is taken, then the resulting algorithm is complete.*

Similarly to the previous result, by increasing the size of large recorded clauses or the life-span of large recorded clauses each time an AFB step is taken, we are guaranteed to eventually keep *all* the clauses required to prevent the repetition of conflict sub-paths, and so either prove satisfiability or unsatisfiability. Observe that for this last result the number of clauses can grow exponentially with the number of variables.

We should note that the observation regarding increasing the relevance-based learning threshold was first suggested in (Moskewicz *et al.* 2001).

One final result addresses the number of times conflict paths and conflict sub-paths can be repeated.

Corollary 13 *Under the conditions of Theorem 11 and Theorem 12, the number of times a a conflict path or a conflict sub-path is repeated is upper-bounded.*

Clearly, the reasoning that was used for establishing Corollary 10 can also be applied in this case.

Related Work

Different variations of non-chronological backtracking (e.g. backjumping) and different forms of nogood learning were originally proposed by Stallman and Sussman in (Stallman & Sussman 1977) in the area of Truth Maintenance Systems (TMS), and independently studied by J. Gaschnig (Gaschnig 1979) and others (see for example (Dechter 1990) in the context of Constraint Satisfaction Problems (CSP)). Moreover, a thorough analysis of conflict-directed backjumping can be found in (Chen & van Beek 2001).

The introduction of variations in the backtrack step is also related to dynamic backtracking (Ginsberg 1993). Dynamic backtracking establishes a method by which backtrack points can be moved deeper in the search tree. This allows avoiding the unneeded erasing of the amount of search that has been done thus far. The target is to find a way to directly "erase" the value assigned to a variable as opposed to backtracking to it, moving the backjump variable to the end of the partial solution in order to replace its value without modifying the values of the variables that currently follow it. More recently, Ginsberg and McAllester combined local search and dynamic backtracking in an algorithm which enables arbitrary search movement (Ginsberg & McAllester 1994), starting with *any complete assignment* and evolving by flipping values of variables obtained from the conflicts.

In weak-commitment search (Yokoo 1994), the algorithm constructs a consistent partial solution, but commits to the partial solution *weakly*, in contrast to standard backtracking algorithms which never abandon a partial solution unless it turns out to be hopeless (i.e. when it is shown not to yield a solution).

Moreover, search restarts have been proposed and shown effective for real-world instances of SAT (Gomes, Selman, & Kautz 1998). The search is repeatedly restarted whenever a cutoff value is reached. The algorithm proposed is not complete, since the restart cutoff point is kept constant. In (Baptista & Marques-Silva 2000), search restarts were jointly used with learning for solving hard real-world instances of SAT. This later algorithm is complete, since the backtrack cutoff value increases after each restart. More recently, a highly-optimized complete SAT solver (Moskewicz *et al.* 2001) has successfully combined non-chronological backtracking and restarts, again obtaining remarkable results on solving real-world instances of SAT.

Conclusions and Future Work

This paper proposes and analyzes a general framework for unrestrictedly backtracking in SAT algorithms. Moreover, we propose different conditions for ensuring the completeness of SAT algorithms.

In the near future, we expect to study other variations of this new backtracking strategy. We can envision the implementation of different hybrids (e.g. weak-commitment search, search restarts and random backtracking), all guaranteed to be complete and so capable of proving unsatisfiability. In this generic framework, the actual backtrack point can be defined using either randomization, heuristic knowledge, constant-depth backtracking or search restarts, among other possible approaches. In addition, it will be important to conduct a comprehensive experimental evaluation and categorization of the proposed backtracking strategies.

References

Baptista, L., and Marques-Silva, J. P. 2000. Using randomization and learning to solve hard real-world instances of satisfiability. In *International Conference on Principles and Practice of Constraint Programming*, 489–494.

Bayardo Jr., R., and Schrag, R. 1997. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the National Conference on Artificial Intelligence*, 203–208.

Chen, X., and van Beek, P. 2001. Conflict-directed backjumping revisited. *Journal of Artificial Intelligence Research* 14:53–81.

Davis, M., and Putnam, H. 1960. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery* 7:201–215.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem-proving. *Communications of the Association for Computing Machinery* 5:394–397.

Dechter, R. 1990. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence* 41(3):273–312.

Gaschnig, J. 1979. *Performance Measurement and Analysis of Certain Search Algorithms*. Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, PA.

Ginsberg, M., and McAllester, D. 1994. GSAT and dynamic backtracking. In *Proceedings of the International Conference on Principles of Knowledge and Reasoning*, 226–237.

Ginsberg, M. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research* 1:25–46.

Gomes, C. P.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *Proceedings of the National Conference on Artificial Intelligence*.

Lynce, I.; Baptista, L.; and Marques-Silva, J. 2001. Stochastic systematic search algorithms for satisfiability. In *LICS Workshop on Theory and Applications of Satisfiability Testing*.

Marques-Silva, J. P., and Sakallah, K. A. 1999. GRASP-A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* 48(5):506–521.

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*.

Selman, B., and Kautz, H. 1993. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 290–295.

Stallman, R. M., and Sussman, G. J. 1977. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence* 9:135–196.

Yokoo, M. 1994. Weak-commitment search for solving satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence*, 313–318.

Zhang, H. 1997. SATO: An efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction*, 272–275.